

main.py

```
1  from utils import *
2
3  from composer2d import Axes, Cage, Net, Square
4  from composer3d import Cube, Tetrahedron, Octahedron, Icosahedron, Dodecahedron
5
6  from app import Vertex, Object, App
7
8
9  if __name__ == '__main__':
10     app = App()
11
12     app.add_debug_object(Object(_position=Vertex(0, 0, 0), _shape=Cage()))
13     # app.add_debug_object(Object(_position=Vertex(0, 0, 0), _shape=Net()))
14
15     app.add_object(
16         Object(
17             _position=Vertex(1, 1, 1),
18             _face=Vertex(0, 1, 0),
19             _shape=Cube(Vertex(0, 0, 0), 1, COLORS['red']),
20             _edge=1
21         )
22     )
23     app.add_object(
24         Object(
25             _position=Vertex(-1, 0, 0),
26             _face=Vertex(0, 1, 0),
27             _shape=Tetrahedron(Vertex(0, 0, 0), 1.2, COLORS['green']),
28             _edge=1.2
29         )
30     )
31     app.add_object(
32         Object(
33             _position=Vertex(-2, 0, 0),
34             _face=Vertex(0, 1, 0),
35             _shape=Octahedron(Vertex(0, 0, 0), .9, COLORS['cyan']),
36             _edge=.9
37         )
38     )
39     app.add_object(
40         Object(
41             _position=Vertex(1, 0, 0),
42             _face=Vertex(0, 1, 0),
43             _shape=Icosahedron(Vertex(0, 0, 0), .75, COLORS['yellow_c']),
44             _edge=.75
45         )
46     )
47     app.add_object(
48         Object(
49             _position=Vertex(2, 0, 0),
50             _face=Vertex(0, 1, 0),
51             _shape=Dodecahedron(Vertex(0, 0, 0), .65, COLORS['magenta']),
52             _edge=.65
53         )
54     )
55
56     for shape in app.SHAPES:
```

```
57 |         shape.push()  
58 |         shape.spin()  
59 |  
60 |     app.run(_debug=True)
```

app.py

```
1  from utils import *
2
3  from object import Vertex, Shape2D, Shape3D, Object
4
5  class App:
6      """
7      Main class of the project. Initilising window, lights, manages objects and draw them.
8      """
9      def __init__(self, _resolution : tuple[int, int] = DEFAULT_RESOLUTION) -> None:
10         pygame.init()
11         pygame.display.set_caption('Plato Solids')
12         try:
13             pygame.display.set_icon(pygame.image.load('./rsc/icosahedron.png'))
14         except:
15             try:
16                 pygame.display.set_icon(pygame.image.load('./icosahedron.png'))
17             except:
18                 pass
19         self.display = _resolution
20         self.window = pygame.display.set_mode(self.display, DOUBLEBUF | OPENGGL)
21         gluPerspective(45, (self.display[0] / self.display[1]), 0.1, 50.0)
22         glTranslatef(*CAM_POSITION)
23
24         glLight(GL_LIGHT0, GL_POSITION, (4, 4, 8, 1)) # point light from the left, top,
front
25         glLightfv(GL_LIGHT0, GL_AMBIENT, (0, 0, 0, 1))
26         glLightfv(GL_LIGHT0, GL_DIFFUSE, (1, 1, 1, 1))
27
28         glEnable(GL_DEPTH_TEST)
29         # glDepthFunc(GL_LESS)
30
31         self.SHAPES_DEBUG : list[Object] = list()
32         self.SHAPES : list[Object] = list()
33
34     def add_debug_object(self, _object : Object) -> None:
35         """
36         Adds new objects, that are used as debug .
37         """
38         self.SHAPES_DEBUG.append(_object)
39
40     def add_object(self, _object : Object) -> None:
41         """
42         Adds new objects, that will be drawn.
43         """
44         self.SHAPES.append(_object)
45
46     def draw_objects(self, _debug):
47         """
48         It draws all the objects you added .
49         """
50
51         if _debug:
52             for shape in self.SHAPES_DEBUG:
53                 shape.draw(True, False)
54
55         for shape in self.SHAPES:
```

```

56
57     shape.draw(True, True)
58     shape.translate()
59     shape.rotate()
60     shape.collide()
61
62     for _shape in self.SHAPES:
63         if shape is not _shape:
64             # print()
65             def lst_mul(lst, x):
66                 return [a * x for a in lst]
67             if shape.position.distance(_shape.position) <= (shape.edge +
_shape.edge) * .7:
68                 shape.translation, _shape.translation = lst_mul(shape.translation,
-1), lst_mul(_shape.translation, -1)
69
70         # print(shape)
71
72 def run(self, _debug : bool = False):
73     """
74     Runs project. You can rotate cam Up/Down and Left/Right.
75     """
76     clock = pygame.time.Clock()
77     is_over = False
78
79     global FPS_COUNT
80
81     while not is_over:
82         dX = 0
83         dY = 0
84         dZ = 0
85         angle = 0
86         for event in pygame.event.get():
87             if event.type == pygame.QUIT or event.type == pygame.K_ESCAPE:
88                 is_over = True
89                 pygame.quit()
90                 quit()
91             if event.type == pygame.KEYDOWN:
92                 if event.key == pygame.K_UP:
93                     dX += 1
94                     angle = 15
95                 if event.key == pygame.K_DOWN:
96                     dX -= 1
97                     angle = 15
98
99                 if event.key == pygame.K_LEFT:
100                     dY += 1
101                     angle = 15
102                 if event.key == pygame.K_RIGHT:
103                     dY -= 1
104                     angle = 15
105
106                 if event.key == pygame.K_KP7:
107                     dZ += 1
108                     angle = 15
109                 if event.key == pygame.K_KP9:
110                     dZ -= 1
111                     angle = 15
112
113     glRotatef(angle, dX, dY, dZ)

```

```
114         # glRotatef(.5, 1, 1, 1)
115
116         FPS_COUNT += 1
117
118         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
119
120         glEnable(GL_LIGHT0)
121         glEnable(GL_LIGHTING)
122         glEnable(GL_COLOR_MATERIAL)
123
124         self.draw_objects(_debug)
125
126         glDisable(GL_LIGHT0)
127         glDisable(GL_LIGHTING)
128         glDisable(GL_COLOR_MATERIAL)
129
130         clock.tick(FPS_CAP)
131         # pygame.display.set_caption(f'{int(clock.get_fps())} [{FPS_COUNT}]')
132         pygame.display.flip()
133
134         # pygame.time.wait(30)
135         # sleep(2)
136
137 if __name__ == '__main__':
138     pass
```

object.py

```
1  from utils import *
2  from shape3d import Vertex, Shape2D, Shape3D
3
4  class Object:
5      def __init__(self,
6          _position : Vertex = Vertex(0, 0, 0),
7          _face : Vertex = Vertex(0, 0, 1),
8          _shape : Shape2D | Shape3D = None,
9          _edge : float = 1
10         ) -> None:
11         self.position : Vertex = _position
12         self.face : Vertex = _face
13         self.translation : list[float] = None
14         self.rotation : tuple[float, str] = None
15         self.shape : Shape2D | Shape3D = _shape
16         self.edge : float = _edge
17
18     def __repr__(self) -> str:
19         return f"POS: {self.position}\npos: {self.shape.position}\nFACE: {self.face}\nT:
{as_colored(self.translation, fg=FOREGROUND_COLORS_STRONG['yellow'])}\nROT:
{as_colored(self.rotation, fg=FOREGROUND_COLORS_STRONG['red'])}\n{self.shape.position -
self.shape.vertices[0]}"
20
21     def push(self) -> None:
22         """
23         Give a random uniform vector of traslation
24         """
25         self.translation = [
26             uniform(*UNIFORM_DIRECTION_AREA),
27             uniform(*UNIFORM_DIRECTION_AREA),
28             uniform(*UNIFORM_DIRECTION_AREA)
29         ]
30
31     def add_translation(self, _vector : list[float]) -> None:
32         self.translation = _vector
33
34     def add_rotation(self, _angle : float, _axis : str) -> None:
35         self.rotation = (_angle, _axis)
36
37     def draw_vector(self, _vector):
38         glLineWidth(2)
39         glBegin(GL_LINES)
40         glColor3f(*COLORS['cyan'])
41         glVertex3f(self.position.x, self.position.y, self.position.z)
42         glVertex3f(self.position.x + _vector.x, self.position.y + _vector.y,
self.position.z + _vector.z)
43         glEnd()
44         glLineWidth(1)
45
46     def draw_y_axis(self):
47         glLineWidth(2)
48         glBegin(GL_LINES)
49         glColor3f(*COLORS['red'])
50         glVertex3f(self.position.x, self.position.y+1, self.position.z)
51         glVertex3f(self.position.x, self.position.y-1, self.position.z)
52         glEnd()
53         glLineWidth(1)
```

```

54
55     def draw_center(self):
56         glPointSize(8)
57         glBegin(GL_POINTS)
58         glColor3f(*COLORS['white'])
59         glVertex3f(self.position.x, self.position.y, self.position.z)
60         glEnd()
61         glPointSize(1)
62
63         glPointSize(8)
64         glBegin(GL_POINTS)
65         glColor3f(*COLORS['cyan'])
66         glVertex3f(self.position.x + self.shape.position.x, self.position.y +
self.shape.position.y, self.position.z + self.shape.position.z)
67         glEnd()
68         glPointSize(1)
69
70     def draw(self, _draw_edges : bool = False, _draw_vertices : bool = False) -> None:
71
72         self.shape.draw(self.position, _draw_edges, _draw_vertices)
73
74     def translate(self) -> None:
75         if self.translation:
76             self.position.translate(self.translation)
77
78
79     def rotate(self) -> None:
80         """
81         Rotation in relation to Object Origin (Rotating only the object position and face
Vector).
82         """
83         if self.rotation:
84             if len(self.rotation) == 2:
85                 self.face.rotate(Vertex(0, 0, 0), *self.rotation)
86                 # self.position.rotate(self.position, *self.rotation)
87                 self.shape.rotate(Vertex(0, 0, 0), *self.rotation)
88             elif len(self.rotation) == 3:
89                 self.face.rotate_any(Vertex(0, 0, 0), *self.rotation)
90                 # self.position.rotate_any(self.position, *self.rotation)
91                 self.shape.rotate_any(Vertex(0, 0, 0), *self.rotation)
92
93
94     def spin(self):
95         self.rotation = [
96             uniform(*UNIFORM_DIRECTION_AREA),
97             uniform(*UNIFORM_DIRECTION_AREA),
98             uniform(*UNIFORM_DIRECTION_AREA)
99         ]
100
101     def collide(self):
102         if self.translation:
103             collision = [1, 1, 1]
104             if self.position.x > BORDER_COLLITION_CAP or self.position.x < -
BORDER_COLLITION_CAP:
105                 collision[0] = -1
106             if self.position.y > BORDER_COLLITION_CAP or self.position.y < -
BORDER_COLLITION_CAP:
107                 collision[1] = -1
108             if self.position.z > BORDER_COLLITION_CAP or self.position.z < -
BORDER_COLLITION_CAP:
109                 collision[2] = -1

```

```
110
111         self.translation = [
112             self.translation[0] * collision[0],
113             self.translation[1] * collision[1],
114             self.translation[2] * collision[2]]
115
116
117 if __name__ == '__main__':
118     pass
119
```


shape3d.py

```
1  from utils import *
2  from shape2d import Vertex, Shape2D
3
4  class Shape3D:
5      def __init__(self, _position : Vertex = Vertex(0, 0, 0), _vertices : list[Vertex] =
        list(), _surfaces : list[Shape2D] = list(), _color : tuple[float] = None, _direction :
        list[float] = None) -> None:
6          self.position : Vertex = _position
7          self.vertices : list[Vertex] = _vertices
8          self.surfaces : list[Shape2D] = _surfaces
9          self.color : tuple[float] = _color
10         self.direction : list[int] = [uniform(*UNIFORM_DIRECTION_AREA),
uniform(*UNIFORM_DIRECTION_AREA), uniform(*UNIFORM_DIRECTION_AREA)] if not _direction else
        _direction
11
12         def darken_color(self, _color, _delta = 0.2):
13             return (_color[0] - _delta, _color[1] - _delta, _color[2] - _delta)
14
15         def draw(self, _object_position : Vertex, _draw_edges : bool = False, _draw_vertices :
        bool = False) -> None:
16             for surface in self.surfaces:
17                 if _draw_edges:
18                     for i in range(len(surface.vertices) - 1):
19                         glLineWidth(2)
20                         glBegin(GL_LINES)
21                         glColor3f(*self.darken_color(surface.color))
22                         glVertex3f(
23                             _object_position.x + surface.vertices[i].x,
24                             _object_position.y + surface.vertices[i].y,
25                             _object_position.z + surface.vertices[i].z
26                         )
27                         glVertex3f(
28                             _object_position.x + surface.vertices[i + 1].x,
29                             _object_position.y + surface.vertices[i + 1].y,
30                             _object_position.z + surface.vertices[i + 1].z
31                         )
32                         glEnd()
33                         glLineWidth(1)
34                 surface.draw(_object_position, _draw_vertices)
35
36         def translate(self, _vector : list[float]) -> None:
37             for vertex in self.vertices:
38                 vertex.translate(_vector)
39             self.position.translate(_vector)
40
41         def rotate(self, _object_position : Vertex, _angle : float, _axis : str) -> None:
42             for vertex in self.vertices:
43                 vertex.rotate(_object_position, _angle, _axis)
44             self.position.rotate(_object_position, _angle, _axis)
45
46
47         def rotate_any(self, _object_position : Vertex, _yaw : float, _pitch : float, _roll :
        float) -> None:
48             for vertex in self.vertices:
49                 vertex.rotate_any(_object_position, _yaw, _pitch, _roll)
50             self.position.rotate_any(_object_position, _yaw, _pitch, _roll)
51
```

composer3d.py

```
1  from utils import *
2  from shape3d import Vertex, Shape2D, Shape3D
3
4  def Cube(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color : tuple[float] =
None):
5      """
6      Initialize 3D Cube object .
7      """
8      half_edge = _edge / 2
9      ...
10         3      2
11         0      1
12
13         7      6
14         4      5
15     ...
16     VERTICES = [
17         Vertex(
18             _position.x - half_edge,
19             _position.y - half_edge,
20             _position.z - half_edge,
21         ),
22         Vertex(
23             _position.x - half_edge,
24             _position.y + half_edge,
25             _position.z - half_edge,
26         ),
27         Vertex(
28             _position.x + half_edge,
29             _position.y + half_edge,
30             _position.z - half_edge,
31         ),
32         Vertex(
33             _position.x + half_edge,
34             _position.y - half_edge,
35             _position.z - half_edge,
36         ),
37
38         Vertex(
39             _position.x - half_edge,
40             _position.y - half_edge,
41             _position.z + half_edge,
42         ),
43         Vertex(
44             _position.x - half_edge,
45             _position.y + half_edge,
46             _position.z + half_edge,
47         ),
48         Vertex(
49             _position.x + half_edge,
50             _position.y + half_edge,
51             _position.z + half_edge,
52         ),
53         Vertex(
54             _position.x + half_edge,
55             _position.y - half_edge,
```

```

56         _position.z + half_edge,
57     ),
58
59 ]
60     return Shape3D(_position, VERTICES, [
61         Shape2D(_position, [
62             VERTICES[0],
63             VERTICES[1],
64             VERTICES[2],
65             VERTICES[3],
66         ], _color),
67         Shape2D(_position, [
68             VERTICES[4],
69             VERTICES[5],
70             VERTICES[6],
71             VERTICES[7],
72         ], _color),
73
74         Shape2D(_position, [
75             VERTICES[0],
76             VERTICES[1],
77             VERTICES[5],
78             VERTICES[4],
79         ], _color),
80         Shape2D(_position, [
81             VERTICES[2],
82             VERTICES[3],
83             VERTICES[7],
84             VERTICES[6],
85         ], _color),
86
87         Shape2D(_position, [
88             VERTICES[0],
89             VERTICES[3],
90             VERTICES[7],
91             VERTICES[4],
92         ], _color),
93         Shape2D(_position, [
94             VERTICES[1],
95             VERTICES[2],
96             VERTICES[6],
97             VERTICES[5],
98         ], _color),
99     ])
100
101
102
103 def Tetrahedron(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color :
tuple[float] = None):
104     """
105     Initialize 3D Tetrahedron object .
106     """
107     half_edge = _edge / 2
108     y_base = _edge / 4 * ((2 / 3) ** (1 / 2))
109     z_diff = _edge / (2 * (3 ** (1 / 2)))
110     '''
111         3
112
113         2
114     0      1

```

```

115     ...
116     VERTICES = [
117         Vertex(
118             _position.x - half_edge,
119             _position.y - y_base,
120             _position.z - z_diff,
121         ),
122         Vertex(
123             _position.x + half_edge,
124             _position.y - y_base,
125             _position.z - z_diff,
126         ),
127         Vertex(
128             _position.x,
129             _position.y - y_base,
130             _position.z + 2 * z_diff,
131         ),
132         Vertex(
133             _position.x,
134             _position.y + 3 * y_base,
135             _position.z,
136         ),
137     ]
138 ]
139 return Shape3D(_position, VERTICES, [
140     Shape2D(_position, [
141         VERTICES[0],
142         VERTICES[1],
143         VERTICES[2],
144     ], _color),
145     Shape2D(_position, [
146         VERTICES[0],
147         VERTICES[1],
148         VERTICES[3],
149     ], _color),
150     Shape2D(_position, [
151         VERTICES[1],
152         VERTICES[2],
153         VERTICES[3],
154     ], _color),
155     Shape2D(_position, [
156         VERTICES[2],
157         VERTICES[0],
158         VERTICES[3],
159     ], _color),
160 ])
161
162
163
164 def Octahedron(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color :
tuple[float] = None):
165     """
166     Initialize 3D Octahedron object .
167     """
168     half_edge = _edge / 2
169     height = _edge * ((1 / 2) ** (1 / 2))
170     ...
171         4
172
173         3             2

```

```

174         0         1
175
176         5
177     ...
178     VERTICES = [
179         Vertex(
180             _position.x - half_edge,
181             _position.y,
182             _position.z - half_edge,
183         ),
184         Vertex(
185             _position.x + half_edge,
186             _position.y,
187             _position.z - half_edge,
188         ),
189         Vertex(
190             _position.x + half_edge,
191             _position.y,
192             _position.z + half_edge,
193         ),
194         Vertex(
195             _position.x - half_edge,
196             _position.y,
197             _position.z + half_edge,
198         ),
199         Vertex(
200             _position.x,
201             _position.y + height,
202             _position.z,
203         ),
204         Vertex(
205             _position.x,
206             _position.y - height,
207             _position.z,
208         ),
209
210     ]
211     return Shape3D(_position, VERTICES, [
212         Shape2D(_position, [
213             VERTICES[0],
214             VERTICES[1],
215             VERTICES[4],
216         ], _color),
217         Shape2D(_position, [
218             VERTICES[0],
219             VERTICES[1],
220             VERTICES[5],
221         ], _color),
222
223         Shape2D(_position, [
224             VERTICES[1],
225             VERTICES[2],
226             VERTICES[4],
227         ], _color),
228         Shape2D(_position, [
229             VERTICES[1],
230             VERTICES[2],
231             VERTICES[5],
232         ], _color),
233

```

```

234     Shape2D(_position, [
235         VERTICES[2],
236         VERTICES[3],
237         VERTICES[4],
238     ], _color),
239     Shape2D(_position, [
240         VERTICES[2],
241         VERTICES[3],
242         VERTICES[5],
243     ], _color),
244
245     Shape2D(_position, [
246         VERTICES[3],
247         VERTICES[0],
248         VERTICES[4],
249     ], _color),
250     Shape2D(_position, [
251         VERTICES[3],
252         VERTICES[0],
253         VERTICES[5],
254     ], _color),
255
256 ])
257
258
259 def Icosahedron(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color :
tuple[float] = None):
260     """
261     Initialize 3D Icosahedron object .
262     """
263     fi = pi * 2 / 5
264     r = _edge / (2 * sin(fi / 2))
265     y_top = _edge / 2 * (3 - 1 / sin(fi / 2)) ** (1 / 2)
266     circumradius = ((10 + (2 * 5 ** (1 / 2))) ** (1 / 2)) * (_edge / 4)
267     print(circumradius, y_top)
268     '''
269         5
270         3
271     4         2
272         0         1
273
274         6
275     7         10
276         8         9
277         11
278
279     '''
280     VERTICES = list()
281     for i in range(5):
282         vertex = Vertex(
283             _position.x,
284             _position.y + circumradius - y_top,
285             _position.z - r,
286         )
287         vertex.rotate(_position, fi * i, 'Ro_y')
288         VERTICES.append(vertex)
289     VERTICES.append(
290         Vertex(
291             _position.x,
292             _position.y + circumradius,

```

```

293         _position.z,
294     ))
295
296
297     for i in range(5):
298         vertex = Vertex(
299             _position.x,
300             _position.y - circumradius + y_top,
301             _position.z + r,
302         )
303         vertex.rotate(_position, fi * i, 'Ro_y')
304         VERTICES.append(vertex)
305     VERTICES.append(
306         Vertex(
307             _position.x,
308             _position.y - circumradius,
309             _position.z,
310         ))
311
312     return Shape3D(_position, VERTICES, [
313         Shape2D(_position, [
314             VERTICES[0],
315             VERTICES[1],
316             VERTICES[5],
317         ], _color),
318         Shape2D(_position, [
319             VERTICES[1],
320             VERTICES[2],
321             VERTICES[5],
322         ], _color),
323         Shape2D(_position, [
324             VERTICES[2],
325             VERTICES[3],
326             VERTICES[5],
327         ], _color),
328         Shape2D(_position, [
329             VERTICES[3],
330             VERTICES[4],
331             VERTICES[5],
332         ], _color),
333         Shape2D(_position, [
334             VERTICES[4],
335             VERTICES[0],
336             VERTICES[5],
337         ], _color),
338
339         Shape2D(_position, [
340             VERTICES[0],
341             VERTICES[1],
342             VERTICES[9],
343         ], _color),
344         Shape2D(_position, [
345             VERTICES[1],
346             VERTICES[2],
347             VERTICES[10],
348         ], _color),
349         Shape2D(_position, [
350             VERTICES[2],
351             VERTICES[3],
352             VERTICES[6],

```

```
353 ], _color),
354 Shape2D(_position, [
355     VERTICES[3],
356     VERTICES[4],
357     VERTICES[7],
358 ], _color),
359 Shape2D(_position, [
360     VERTICES[4],
361     VERTICES[0],
362     VERTICES[8],
363 ], _color),
364
365 Shape2D(_position, [
366     VERTICES[6],
367     VERTICES[7],
368     VERTICES[3],
369 ], _color),
370 Shape2D(_position, [
371     VERTICES[7],
372     VERTICES[8],
373     VERTICES[4],
374 ], _color),
375 Shape2D(_position, [
376     VERTICES[8],
377     VERTICES[9],
378     VERTICES[0],
379 ], _color),
380 Shape2D(_position, [
381     VERTICES[9],
382     VERTICES[10],
383     VERTICES[1],
384 ], _color),
385 Shape2D(_position, [
386     VERTICES[10],
387     VERTICES[6],
388     VERTICES[2],
389 ], _color),
390
391 Shape2D(_position, [
392     VERTICES[6],
393     VERTICES[7],
394     VERTICES[11],
395 ], _color),
396 Shape2D(_position, [
397     VERTICES[7],
398     VERTICES[8],
399     VERTICES[11],
400 ], _color),
401 Shape2D(_position, [
402     VERTICES[8],
403     VERTICES[9],
404     VERTICES[11],
405 ], _color),
406 Shape2D(_position, [
407     VERTICES[9],
408     VERTICES[10],
409     VERTICES[11],
410 ], _color),
411 Shape2D(_position, [
412     VERTICES[10],
```



```

413         VERTICES[6],
414         VERTICES[11],
415     ], _color),
416 ])
417
418
419 def Dodecahedron(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color :
tuple[float] = None):
420     """
421     Initialize 3D Dodecahedron object .
422     """
423     fi = pi * 2 / 5
424     r = _edge / (2 * sin(fi / 2))
425     circumradius = (3 ** (1 / 2)) * (1 + (5 ** (1 / 2))) * _edge / 4
426     _y_top = _edge / 2 * (3 * (3 + 2 * (5 ** (1 / 2))) / 2 - 1 / sin(fi / 2)) ** (1 / 2)
427
428     y_top = (circumradius ** 2 - r ** 2) ** (1 / 2)
429     d = _edge / 2 * (1 + (5 ** (1 / 2)))
430     r_ = d / (2 * sin(fi / 2))
431     y_mid = (circumradius ** 2 - r_ ** 2) ** (1 / 2)
432
433
434     # print(circumradius, y_top)
435     '''
436
437         3
438     4      2
439     0      1
440
441     8      7
442     9      6
443         5
444
445     '''
446     VERTICES = list()
447
448     # top
449     for i in range(5):
450         vertex = Vertex(
451             _position.x,
452             _position.y + y_top,
453             _position.z - r,
454         )
455         vertex.rotate(_position, fi * i, 'Ro_y')
456         VERTICES.append(vertex)
457
458     # top-mid
459     for i in range(5):
460         vertex = Vertex(
461             _position.x,
462             _position.y + y_mid,
463             _position.z - r_,
464         )
465         vertex.rotate(_position, fi * i, 'Ro_y')
466         VERTICES.append(vertex)
467
468     # bottom-mid
469     for i in range(5):
470         vertex = Vertex(

```

```

472         _position.x,
473         _position.y - y_mid,
474         _position.z + r_,
475     )
476     vertex.rotate(_position, fi * i, 'Ro_y')
477     VERTICES.append(vertex)
478
479 # bottom
480 for i in range(5):
481     vertex = Vertex(
482         _position.x,
483         _position.y - y_top,
484         _position.z + r,
485     )
486     vertex.rotate(_position, fi * i, 'Ro_y')
487     VERTICES.append(vertex)
488
489 return Shape3D(_position, VERTICES, [
490     Shape2D(_position, [
491         VERTICES[0],
492         VERTICES[1],
493         VERTICES[2],
494         VERTICES[3],
495         VERTICES[4],
496     ], _color),
497
498     Shape2D(_position, [
499         VERTICES[0],
500         VERTICES[1],
501         VERTICES[6],
502         VERTICES[-7],
503         VERTICES[5],
504     ], _color),
505     Shape2D(_position, [
506         VERTICES[1],
507         VERTICES[2],
508         VERTICES[7],
509         VERTICES[-6],
510         VERTICES[6],
511     ], _color),
512     Shape2D(_position, [
513         VERTICES[2],
514         VERTICES[3],
515         VERTICES[8],
516         VERTICES[-10],
517         VERTICES[7],
518     ], _color),
519     Shape2D(_position, [
520         VERTICES[3],
521         VERTICES[4],
522         VERTICES[9],
523         VERTICES[-9],
524         VERTICES[8],
525     ], _color),
526     Shape2D(_position, [
527         VERTICES[4],
528         VERTICES[0],
529         VERTICES[5],
530         VERTICES[-8],
531         VERTICES[9],

```

```

532         ], _color),
533
534         Shape2D(_position, [
535             VERTICES[-1],
536             VERTICES[-2],
537             VERTICES[-7],
538             VERTICES[6],
539             VERTICES[-6],
540         ], _color),
541         Shape2D(_position, [
542             VERTICES[-2],
543             VERTICES[-3],
544             VERTICES[-8],
545             VERTICES[5],
546             VERTICES[-7],
547         ], _color),
548         Shape2D(_position, [
549             VERTICES[-3],
550             VERTICES[-4],
551             VERTICES[-9],
552             VERTICES[9],
553             VERTICES[-8],
554         ], _color),
555         Shape2D(_position, [
556             VERTICES[-4],
557             VERTICES[-5],
558             VERTICES[-10],
559             VERTICES[8],
560             VERTICES[-9],
561         ], _color),
562         Shape2D(_position, [
563             VERTICES[-5],
564             VERTICES[-1],
565             VERTICES[-6],
566             VERTICES[7],
567             VERTICES[-10],
568         ], _color),
569
570         Shape2D(_position, [
571             VERTICES[-1],
572             VERTICES[-2],
573             VERTICES[-3],
574             VERTICES[-4],
575             VERTICES[-5],
576         ], _color),
577     ])
578

```

shape2d.py

```
1 from vertex import Vertex
2 from utils import *
3
4 class Shape2D:
5     def __init__(self, _position : Vertex = Vertex(0, 0, 0), _vertices : list[Vertex] =
6 list(), _color : tuple[float] = None, _render_mode = GL_TRIANGLE_FAN) -> None:
7         self.position : Vertex = _position
8         self.vertices : list[Vertex] = _vertices
9         self.color = _color
10        self.render_mode = _render_mode
11
12    def draw(self, _object_position : Vertex, _true, _draw_vertices : bool = False) ->
13None:
14        glBegin(self.render_mode)
15        glColor3f(*self.color)
16        for vertex in self.vertices:
17            glVertex3f(
18                _object_position.x + vertex.x,
19                _object_position.y + vertex.y,
20                _object_position.z + vertex.z)
21            if _draw_vertices:
22                vertex.draw()
23        glEnd()
24
25    def translate(self, _vector : list[float]) -> None:
26        for vertex in self.vertices:
27            vertex.translate(_vector)
28
29    def rotate(self, _object_position : Vertex, _angle : float, _axis : str) -> None:
30        for vertex in self.vertices:
31            vertex.rotate(_object_position, _angle, _axis)
32
33    def rotate_any(self, _object_position : Vertex, _yaw : float, _pitch : float, _roll :
34float) -> None:
35        for vertex in self.vertices:
36            vertex.rotate_any(_object_position, _yaw, _pitch, _roll)
```

composer2d.py

```
1  from utils import *
2  from shape2d import Vertex, Shape2D
3
4  def Line(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color : tuple[float] =
  None, _rotation : tuple[float, str] = None):
5      """
6      Initialize 2D Line object .
7      """
8      half_edge = _edge / 2
9      VERTICES = [
10         Vertex(
11             _position.x,
12             _position.y,
13             _position.z - half_edge,
14         ),
15         Vertex(
16             _position.x,
17             _position.y,
18             _position.z + half_edge,
19         ),
20     ]
21
22     return Shape2D(_position, VERTICES, _color, GL_LINES)
23
24
25 def Axes():
26     """
27     Initialize 2D Axes object .
28     """
29     half_edge = 4
30     VERTICES = [
31         Vertex(
32             0,
33             0,
34             0,
35         ),
36         Vertex(
37             0,
38             0,
39             -half_edge,
40         ),
41
42         Vertex(
43             0,
44             0,
45             0,
46         ),
47         Vertex(
48             0,
49             0,
50             half_edge,
51         ),
52
53         Vertex(
54             0,
55             0,
```

```

56         0,
57     ),
58     Vertex(
59         0,
60         -half_edge,
61         0,
62     ),
63
64     Vertex(
65         0,
66         0,
67         0,
68     ),
69     Vertex(
70         0,
71         half_edge,
72         0,
73     ),
74
75     Vertex(
76         0,
77         0,
78         0,
79     ),
80     Vertex(
81         -half_edge,
82         0,
83         0,
84     ),
85
86     Vertex(
87         0,
88         0,
89         0,
90     ),
91     Vertex(
92         half_edge,
93         0,
94         0,
95     ),
96
97 ]
98 return Shape2D(Vertex(0, 0, 0), VERTICES, COLORS['green'], GL_LINES)
99
100
101 def Net(_size : int = 4):
102     """
103     Initialize 2D Net (every 1 unit = line) object .
104     """
105     len = _size * 2
106     VERTICES = []
107     for z in range(-_size, _size+1):
108         for a in range(-_size, _size+1):
109             VERTICES.append(Vertex(a, -_size, z))
110             VERTICES.append(Vertex(a, +_size, z))
111
112             VERTICES.append(Vertex(-_size, a, z))
113             VERTICES.append(Vertex(+_size, a, z))
114
115     for y in range(-_size, _size+1):

```

```

116         for x in range(-_size, _size+1):
117             VERTICES.append(Vertex(x, y, -_size))
118             VERTICES.append(Vertex(x, y, +_size))
119
120     return Shape2D(Vertex(0, 0, 0), VERTICES, COLORS['g_cage'], GL_LINES)
121
122
123 def Cage(_size : int = 4):
124     """
125     Initialize 2D Cage (Borders) object .
126     """
127     VERTICES = [
128         Vertex(
129             -_size,
130             -_size,
131             -_size,
132         ),
133         Vertex(
134             +_size,
135             -_size,
136             -_size,
137         ),
138         Vertex(
139             -_size,
140             -_size,
141             -_size,
142         ),
143         Vertex(
144             -_size,
145             +_size,
146             -_size,
147         ),
148         Vertex(
149             -_size,
150             -_size,
151             -_size,
152         ),
153         Vertex(
154             -_size,
155             -_size,
156             +_size,
157         ),
158
159         Vertex(
160             -_size,
161             +_size,
162             -_size,
163         ),
164         Vertex(
165             -_size,
166             +_size,
167             +_size,
168         ),
169         Vertex(
170             -_size,
171             +_size,
172             -_size,
173         ),
174         Vertex(
175             +_size,

```

```
176         +_size,
177         -_size,
178     ),
179
180     Vertex(
181         +_size,
182         +_size,
183         +_size,
184     ),
185     Vertex(
186         -_size,
187         +_size,
188         +_size,
189     ),
190     Vertex(
191         +_size,
192         +_size,
193         +_size,
194     ),
195     Vertex(
196         +_size,
197         -_size,
198         +_size,
199     ),
200     Vertex(
201         +_size,
202         +_size,
203         +_size,
204     ),
205     Vertex(
206         +_size,
207         +_size,
208         -_size,
209     ),
210
211     Vertex(
212         +_size,
213         -_size,
214         +_size,
215     ),
216     Vertex(
217         -_size,
218         -_size,
219         +_size,
220     ),
221     Vertex(
222         +_size,
223         -_size,
224         +_size,
225     ),
226     Vertex(
227         +_size,
228         -_size,
229         -_size,
230     ),
231
232     Vertex(
233         -_size,
234         -_size,
235         +_size,
```



```

236         ),
237         Vertex(
238             -_size,
239             +_size,
240             +_size,
241         ),
242         Vertex(
243             +_size,
244             -_size,
245             -_size,
246         ),
247         Vertex(
248             +_size,
249             +_size,
250             -_size,
251         ),
252     ]
253
254
255     return Shape2D(Vertex(0, 0, 0), VERTICES, COLORS['g_cage'], GL_LINES)
256
257
258 def Square(_position : Vertex = Vertex(0, 0, 0), _edge : float = 1, _color : tuple[float]
= None, _rotation : tuple[float, str] = None):
259     """
260     Initialize 2D Square object .
261     """
262     half_edge = _edge / 2
263     VERTICES = [
264         Vertex(
265             _position.x - half_edge,
266             _position.y - half_edge,
267             _position.z,
268         ),
269         Vertex(
270             _position.x + half_edge,
271             _position.y - half_edge,
272             _position.z,
273         ),
274         Vertex(
275             _position.x + half_edge,
276             _position.y + half_edge,
277             _position.z,
278         ),
279         Vertex(
280             _position.x - half_edge,
281             _position.y + half_edge,
282             _position.z,
283         )
284     ]
285     square = Shape2D(_position, VERTICES, _color)
286     if _rotation is not None:
287         square.rotate(*_rotation)
288     return square
289
290
291 def Cirlce(_position : Vertex = Vertex(0, 0, 0), _radius : float = 1, _n_sides : int =
100, _color : tuple[float] = None):
292     """
293     Initialize 2D Cirlce object .

```

```
294     """
295     angle = 2 * pi / _n_sides
296     VERTICES = []
297     for i in range(_n_sides):
298         vertex = Vertex(
299             _position.x,
300             _position.y + _radius,
301             _position.z,
302         )
303         vertex.rotate(_position, i * angle, 'Ro_z')
304         VERTICES.append(vertex)
305     return Shape2D(_position, VERTICES, _color)
306
307
308
```

matrix.py

```
1  class Matrix:
2      def __init__(self, _content : list[list[float]] | tuple[float], _is_vector : bool =
False) -> None:
3          if _is_vector:
4              self.V = _content
5              self.m = 1
6              self.n = len(self.V)
7          else:
8              self.M = _content
9              self.m = len(self.M)
10             self.n = len(self.M[0])
11         self.is_vector = _is_vector
12
13     def to_vertex(self) -> tuple[float, float, float]:
14         if self.is_vector:
15             return self.V
16
17     def mul(self, v1, v2):
18         return sum([v1[i] * v2[i] for i in range(len(v1))])
19
20     def T(self):
21         content = [[0 for _ in range(self.m)] for _ in range(self.n)]
22         for m in range(self.m):
23             for n in range(self.n):
24                 content[n][m] = self.M[m][n]
25         return Matrix(content)
26
27
28
29     def __mul__(self, _mul):
30         if self.is_vector and _mul.is_vector and self.n == _mul.n:
31             return self.mul(self.V, _mul.V)
32         elif not self.is_vector and _mul.is_vector:
33             return Matrix(tuple([self.mul(row, _mul.V) for row in self.M]), True)
34         elif not self.is_vector and not _mul.is_vector and self.n == _mul.m:
35             content = []
36             _mul_t = _mul.T()
37             for line_a in self.M:
38                 row = []
39                 for line_b in _mul_t.M:
40                     row.append(self.mul(line_a, line_b))
41                 content.append(row)
42             return Matrix(content)
43
44
45     def __repr__(self) -> str:
46         if self.is_vector:
47             return self.V.__repr__()
48         else:
49             ret = '['\n'
50             for row in self.M:
51                 ret += f'        {row}\n'
52             ret += ']'
53             return ret
54
55
```

```

56 Identity3_Matrix = Matrix(
57     [
58         [1, 0, 0],
59         [0, 1, 0],
60         [0, 0, 1]
61     ]
62 )
63
64 Zero3_Matrix = Matrix(
65     [
66         [0, 0, 0],
67         [0, 0, 0],
68         [0, 0, 0]
69     ]
70 )
71
72 Zero3_Vector = Matrix((0, 0, 0), True)
73
74 from math import cos, sin
75
76 M = {
77     'T'      : lambda _vector : Matrix([
78         [1, 0, 0, _vector[0]],
79         [0, 1, 0, _vector[1]],
80         [0, 0, 1, _vector[2]],
81         [0, 0, 0, 1],
82     ]),
83     'Ro_x'   : lambda _angle : Matrix([
84         [1, 0, 0],
85         [0, cos(_angle), -sin(_angle)],
86         [0, sin(_angle), cos(_angle)],
87     ]),
88     'Ro_y'   : lambda _angle : Matrix([
89
90         [cos(_angle), 0, sin(_angle)],
91         [0, 1, 0],
92         [-sin(_angle), 0, cos(_angle)],
93     ]),
94     'Ro_z'   : lambda _angle : Matrix([
95         [cos(_angle), -sin(_angle), 0],
96         [sin(_angle), cos(_angle), 0],
97         [0, 0, 1],
98     ]),
99     'Sc'     : Zero3_Matrix,
100    'Re_x'    : Zero3_Matrix,
101    'Re_y'    : Zero3_Matrix,
102    'Re_z'    : Zero3_Matrix,
103 }
104
105 def Ro_Any_Matrix(_yaw : float, _pitch : float, _roll : float) -> Matrix:
106     return M['Ro_z'](_yaw) * M['Ro_y'](_pitch) * M['Ro_x'](_roll)
107
108 if __name__ == '__main__':
109     print(Ro_Any_Matrix(1, 1, 1))
110
111

```

vertex.py

```
1 from utils import *
2 from matrix import Matrix, M, Ro_Any_Matrix
3
4 class Vertex:
5     def __init__(self, _x : float, _y : float, _z : float) -> None:
6         self.x : float = _x
7         self.y : float = _y
8         self.z : float = _z
9         self.V : tuple[float, float, float] = self.set_tuple()
10
11     def set_tuple(self) -> tuple[float, float, float]:
12         return (self.x, self.y, self.z)
13
14     def length(self) -> float:
15         return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** (1 / 2)
16
17     def distance(self, _v) -> float:
18         return ((self.x - _v.x) ** 2 + (self.y - _v.y) ** 2 + (self.z - _v.z) ** 2) ** (1 /
19 2)
20
21     def draw(self) -> None:
22         glBegin(GL_POINTS)
23         glVertex3f(self.x, self.y, self.z)
24         glEnd()
25
26     def translate(self, _vector : list[float]) -> list[float]:
27         new_vertex : Matrix = M['T'](_vector) * Matrix([*self.V, 1], _is_vector=True)
28         self.x = new_vertex.V[0]
29         self.y = new_vertex.V[1]
30         self.z = new_vertex.V[2]
31         self.V = new_vertex.V
32
33     def rotate(self, _position, _angle : float, _axis : str) -> None:
34         new_vertex : Matrix = M[_axis](_angle) * Matrix((self - _position).V,
35 _is_vector=True)
36         self.x = new_vertex.V[0] + _position.x
37         self.y = new_vertex.V[1] + _position.y
38         self.z = new_vertex.V[2] + _position.z
39         self.V = new_vertex.V
40
41     def rotate_any(self, _position, _yaw : float, _pitch : float, _roll : float) -> None:
42         new_vertex : Matrix = Ro_Any_Matrix(_yaw, _pitch, _roll) * Matrix((self -
43 _position).V, _is_vector=True)
44         self.x = new_vertex.V[0] + _position.x
45         self.y = new_vertex.V[1] + _position.y
46         self.z = new_vertex.V[2] + _position.z
47         self.V = new_vertex.V
48
49     def __sub__(self, _v):
50         return Vertex(self.x - _v.x, self.y - _v.y, self.z - _v.z)
51
52     def __mul__(self, _mul):
53         if isinstance(_mul, (int, float)):
54             return Vertex(self.x * _mul, self.y * _mul, self.z * _mul)
55
56     def __repr__(self) -> str:
57         return f'({self.x}, {self.y}, {self.z})'
```

console.py

```
1 def clear_console() -> None :
2     from os import system, name
3     if name == 'nt':
4         _ = system('cls')
5     else:
6         _ = system('clear')
7
8 def clear_screen() -> None :
9     print('\x1b[H\x1b[3J', end='')
10    print('\n'*64)
11    print('\x1b[H\x1b[3J', end='')
12
13
14 RESET_COLOR = '0'
15
16 FOREGROUND_COLORS = {
17     'black'      : '30',
18     'red'        : '31',
19     'green'      : '32',
20     'yellow'     : '33',
21     'blue'       : '34',
22     'purple'     : '35',
23     'cyan'       : '36',
24     'white'      : '37'
25 }
26 FOREGROUND_COLORS_STRONG = {
27     'black'      : '90',
28     'red'        : '91',
29     'green'      : '92',
30     'yellow'     : '93',
31     'blue'       : '94',
32     'purple'     : '95',
33     'cyan'       : '96',
34     'white'      : '97'
35 }
36
37 BACKGROUND_COLORS = {
38     'on_black'   : '40',
39     'on_red'     : '41',
40     'on_green'   : '42',
41     'on_yellow'  : '43',
42     'on_blue'    : '44',
43     'on_purple'  : '45',
44     'on_cyan'    : '46',
45     'on_white'   : '47'
46 }
47 BACKGROUND_COLORS_STRONG = {
48     'on_black'   : '100',
49     'on_red'     : '101',
50     'on_green'   : '102',
51     'on_yellow'  : '103',
52     'on_blue'    : '104',
53     'on_purple'  : '105',
54     'on_cyan'    : '106',
55     'on_white'   : '107'
56 }
```

```

57
58 def set_color(fg : str = None, bg : str = None) -> None:
59     if fg:
60         print(f"\x1b[{fg}m", end='')
61     if bg:
62         print(f"\x1b[{bg}m", end='')
63
64 def reset_color() -> None:
65     print(f"\x1b[{RESET_COLOR}m", end='')
66
67 def colored(*msgs : tuple[str], fg : str = None, bg : str = None) -> None:
68     set_color(fg, bg)
69     print(*msgs)
70     reset_color()
71
72 def as_colored(text : str, fg : str = RESET_COLOR, bg : str = RESET_COLOR) -> str :
73     return f"\x1b[{fg}m{bg}m{text}\x1b[0m"
74
75
76 # def error(msg : str) -> None :
77 #     print(f"\x1b[{FOREGROUND_COLORS['red']}m{msg}\x1b[0m")
78
79 def error(*msgs : tuple[str]) -> None:
80     colored(*msgs, fg=FOREGROUND_COLORS_STRONG['red'])
81
82 def as_error(text : str) -> str :
83     return as_colored(text, fg=FOREGROUND_COLORS_STRONG['red'])
84
85 # def alert(msg : str) -> None :
86 #     print(f"\x1b[{FOREGROUND_COLORS['yellow']}m{msg}\x1b[0m")
87
88 def alert(*msgs : tuple[str]) -> None:
89     colored(*msgs, fg=FOREGROUND_COLORS_STRONG['yellow'])
90
91 def as_alert(text : str) -> str :
92     return as_colored(text, fg=FOREGROUND_COLORS_STRONG['yellow'])
93
94
95 if __name__ == '__main__':
96     abc = 123
97     print(abc)
98     print(*[1, 2, 3, 4, 5])
99
100     alert(abc)
101     alert(abc, 'abc')
102
103     error(abc)
104     error(abc, 'abc')
105

```

utils.py

```
1  from math import pi, sin, cos, radians
2  from random import randint, random, uniform
3  from time import sleep
4
5  import pygame
6  from pygame.locals import *
7  from OpenGL.GL import *
8  from OpenGL.GLU import *
9
10 from console import *
11
12 DEFAULT_RESOLUTION = (800, 800)
13 FPS_CAP = 30
14 FPS_COUNT = 0
15 CAM_POSITION = (0, 0, -14)
16 # LIGHT_POSITION = -CAM_POSITION[2]
17 LIGHT_CAP = 4.0
18 LIGHT_POSITION = (0, 0, LIGHT_CAP, 1)
19 LIGHT_DIFFUSION = (LIGHT_CAP, LIGHT_CAP, LIGHT_CAP, 1)
20
21 UNIFORM_DIRECTION_CAP = 0.1
22 UNIFORM_DIRECTION_AREA = (-UNIFORM_DIRECTION_CAP, UNIFORM_DIRECTION_CAP)
23 BORDER_COLLITION_CAP = 4.0
24
25 COLORS = {
26     'white'      : (1, 1, 1),
27     'gray'       : (.75, .75, .75),
28     'dark_gray'  : (.5, .5, .5),
29     'black'      : (0, 0, 0),
30
31     'red'        : (1, 0, 0),
32     'green'      : (0, 1, 0),
33     'g_cage'     : (0, 1, 0),
34     'blue'       : (0, 0, 1),
35
36     'magenta'    : (1, 0, 1),
37     'yellow'     : (1, 1, 0),
38     'yellow_c'   : (1, 0.75, 0),
39     'cyan'       : (0, 1, 1),
40 }
```